Glide(图片加载框架)

集成Picasso 与 Glide

http://blog.csdn.net/guolin_blog/article/details/70215985?utm_source=gold_browser_extension

Glide 系列

Picasso

```
dependencies {
    compile 'com.squareup.picasso:picasso:2.5.1'
}
```

Glide

```
dependencies {
    compile 'com.github.bumptech.glide:glide:3.5.2'
    compile 'com.android.support:support-v4:22.0.0'
}
```

Glide需要依赖**Support Library v4**，别

Glide 与Picasso 最基本加载图片的方法：

Picasso

```
Picasso.with(context)
  .load("http://inthecheesefactory.com/uploads/source/glidepicasso/cover.jpg")
  .into(ivImg);
```

Glide

```
 Glide.with(context)
.load("http://inthecheesefactory.com/uploads/source/glidepicasso/cover.jpg") .into(ivImg);
```

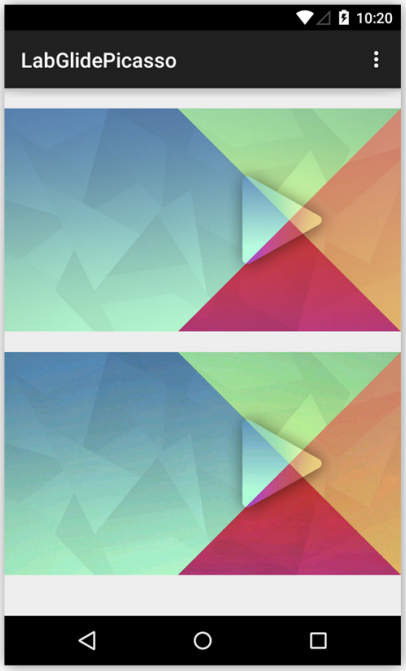
Glide的with方法不光接受Context，还接受Activity 和 Fragment，Context会自动的从他们获取。

同 时将Activity/Fragment作为with()参数的好处是：图片加载会和Activity/Fragment的生命周期保持一致，比如 Paused状态在暂停加载，在Resumed的时候又自动重新加载。所以我建议传参的时候传递Activity 和 Fragment给Glide，而不是Context。

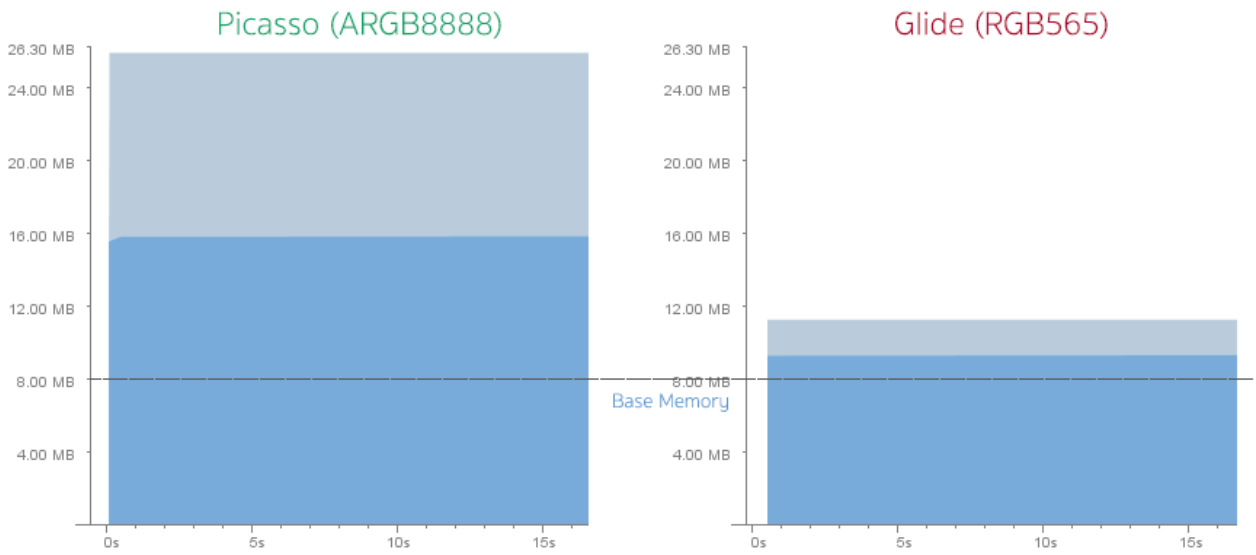
如果传入Context 只有当应用程序被杀掉的时候 图片加载才会停止

默认Bitmap格式是RGB_565

下面是加载图片时和Picasso的比较（1920x1080 像素的图片加载到768x432的ImageView中）



可以看到Glide加载的图片质量要差于Picasso（ps：我看不出来哈），为什么？这是因为Glide默认的Bitmap格式是RGB_565，比ARGB_8888格式的内存开销要小一半。下面是Picasso在ARGB8888下与Glide在RGB565下的内存开销图（应用自身占用了8m，因此以8为基准线比较）：



如果你对默认的RGB_565效果还比较满意，可以不做任何事，但是如果你觉得难以接受，可以创建一个新的GlideModule将Bitmap格式转换到ARGB_8888：

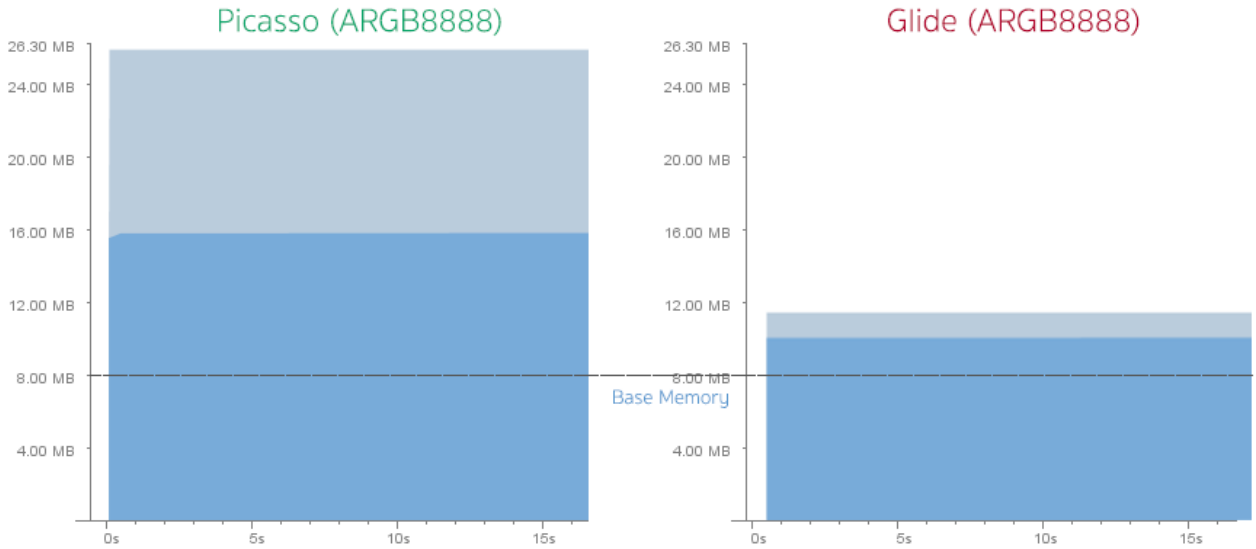
```
public class GlideConfiguration implements GlideModule {

    @Override
    public void applyOptions(Context context, GlideBuilder builder) {
        // Apply options to the builder here.
        builder.setDecodeFormat(DecodeFormat.PREFER_ARGB_8888);
    }

    @Override
    public void registerComponents(Context context, Glide glide) {
        // register ModelLoaders here.
    }
}
```

同时在AndroidManifest.xml中将GlideModule定义为meta-data

```
<meta-data android:name="com.inthecheesefactory.lab.glidepicasso.GlideConfiguration"
        android:value="GlideModule"/>
```

这样看起来就会好很多。

我们再来看看内存开销图，这次貌似Glide花费了两倍于上次的内存，但是Picasso的内存开销仍然远大于Glide。



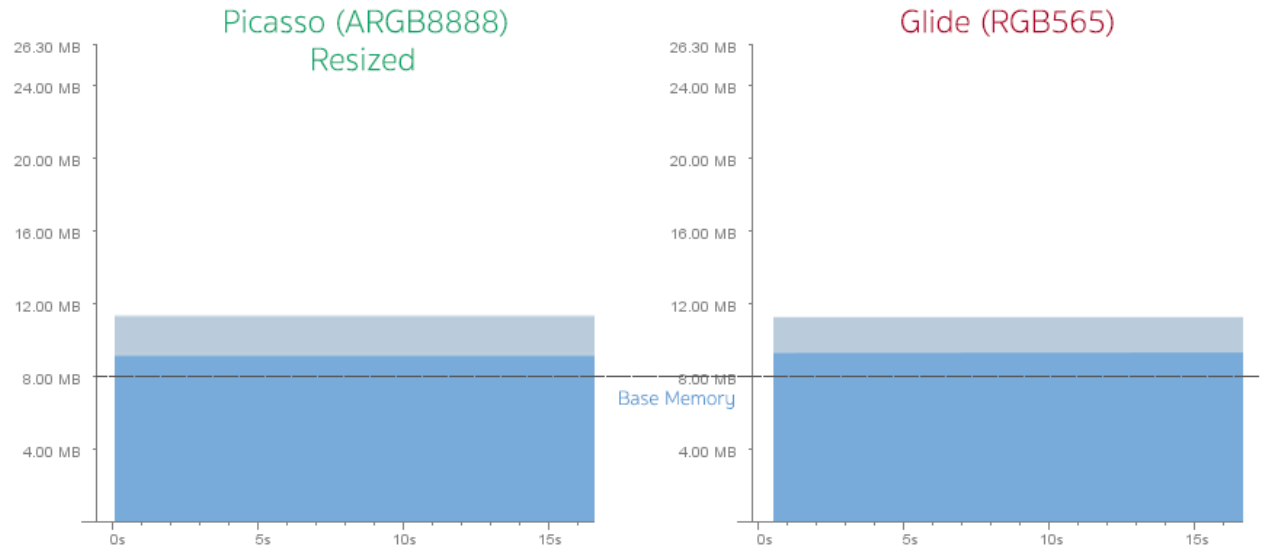
原因在于Picasso是加载了全尺寸的图片到内存，然后让GPU来实时重绘大小。而Glide加载的大小和ImageView的大小是一致的，因此更小。当然，Picasso也可以指定加载的图片大小的：

```
Picasso.with(this)
    .load("http://nuuneoi.com/uploads/source/playstore/cover.jpg")
    .resize(768, 432)
    .into(ivImgPicasso);
```

但是问题在于你需要主动计算ImageView的大小，或者说你的ImageView大小是具体的值（而不是wrap_content），你也可以这样：

```
Picasso.with(this)
    .load("http://nuuneoi.com/uploads/source/playstore/cover.jpg")
    .fit()
    .centerCrop()
    .into(ivImgPicasso);
```

现在Picasso的内存开销就和Glide差不多了。



虽然内存开销差距不到，但是在这个问题上Glide完胜Picasso。因为Glide可以自动计算出任意情况下的ImageView大小。

Image质量的细节

这是将ImageView还原到真实大小时的比较。

你可以看到，Glide加载的图片没有Picasso那么平滑，我还没有找到一个可以直观改变图片大小调整算法的方法。
但是这并不算什么坏事，因为很难察觉。

磁盘缓存

Picasso和Glide在磁盘缓存策略上有很大的不同。Picasso缓存的是全尺寸的，而Glide缓存的是跟ImageView尺寸相同的。

Picasso Cache

Glide Cache

上面提到的平滑度的问题依然存在，而且如果加载的是RGB565图片，那么缓存中的图片也是RGB565。

我 尝试将ImageView调整成不同大小，但不管大小如何Picasso只缓存一个全尺寸的。Glide则不同，它会为每种大小的ImageView缓存 一次。尽管一张图片已经缓存了一次，但是假如你要在另外一个地方再次以不同尺寸显示，需要重新下载，调整成新尺寸的大小，然后将这个尺寸的也缓存起来。

具体说来就是：假如在第一个页面有一个200x200的ImageView，在第二个页面有一个100x100的ImageView，这两个ImageView本来是要显示同一张图片，却需要下载两次。

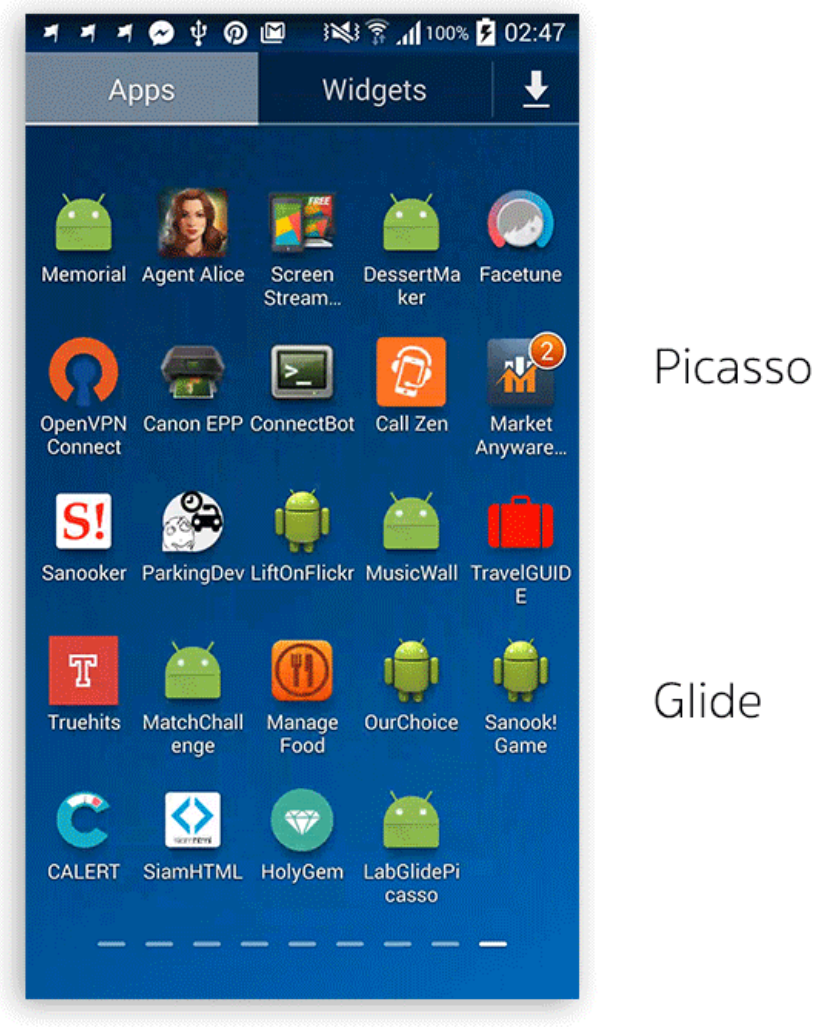
不过，你可以改变这种行为，让Glide既缓存全尺寸又缓存其他尺寸：

```
Glide.with(this)
   .load("http://nuuneoi.com/uploads/source/playstore/cover.jpg")
   .diskCacheStrategy(DiskCacheStrategy.ALL)
   .into(ivImgGlide);
```

下次在任何ImageView中加载图片的时候，全尺寸的图片将从缓存中取出，重新调整大小，然后缓存。

Glide的这种方式优点是加载显示非常快。而Picasso的方式则因为需要在显示之前重新调整大小而导致一些延迟，即便你添加了这段代码来让其立即显示：

1. Picasso
2. .noFade();



Picasso和Glide各有所长，你根据自己的需求选择合适的。
对我而言，我更喜欢Glide，因为它远比Picasso快，虽然需要更大的空间来缓存。

特性

你可以做到几乎和Picasso一样多的事情，代码也几乎一样。

Image Resizing

```
1. // Picasso
2. .resize(300, 200);
3.
4. // Glide
5. .override(300, 200);
```

Center Cropping

```
1. // Picasso
2. .centerCrop();
3.
4. // Glide
5. .centerCrop();
```

Transforming

```
1. // Picasso
2. .transform(new CircleTransform())
```

```
3.
4. // Glide
5. .transform(new CircleTransform(context))
```

设置占位图或者加载错误图：

```
1. // Picasso
2. .placeholder(R.drawable.placeholder)
3. .error(R.drawable.imagenotfound)
4.
5. // Glide
6. .placeholder(R.drawable.placeholder)
7. .error(R.drawable.imagenotfound)
```

几乎和Picasso一样，从Picasso转换到Glide对你来说就是小菜一碟。

有什么Glide可以做而Picasso 做不到

Glide可以加载GIF动态图，而Picasso不能。

同时因为Glide和Activity/Fragment的生命周期是一致的，因此gif的动画也会自动的随着Activity/Fragment的状态暂停、重放。
Glide 的缓存在gif这里也是一样，调整大小然后缓存。
但是从我的一次测试结果来看Glide 动画会消费太多的内存，因此谨慎使用。
除了gif动画之外，Glide还可以将任何的本地视频解码成一张静态图片。
还有一个特性是你可以配置图片显示的动画，而Picasso只有一种动画：fading in。
最后一个是可以使用thumbnail()产生一个你所加载图片的thumbnail。

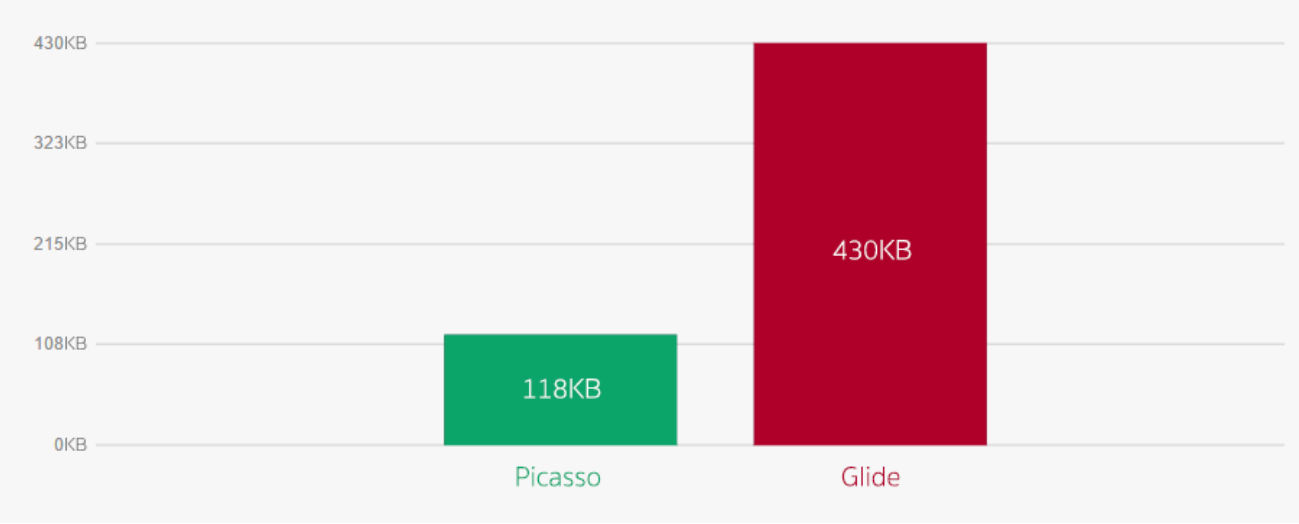
其实还有一些特性，不过不是非常重要，比如将图像转换成字节数组等。

配置

有许多可以配置的选项，比如大小，缓存的磁盘位置，最大缓存空间，位图格式等等。可以在这个页面查看这些配置 Configuration 。

库的大小

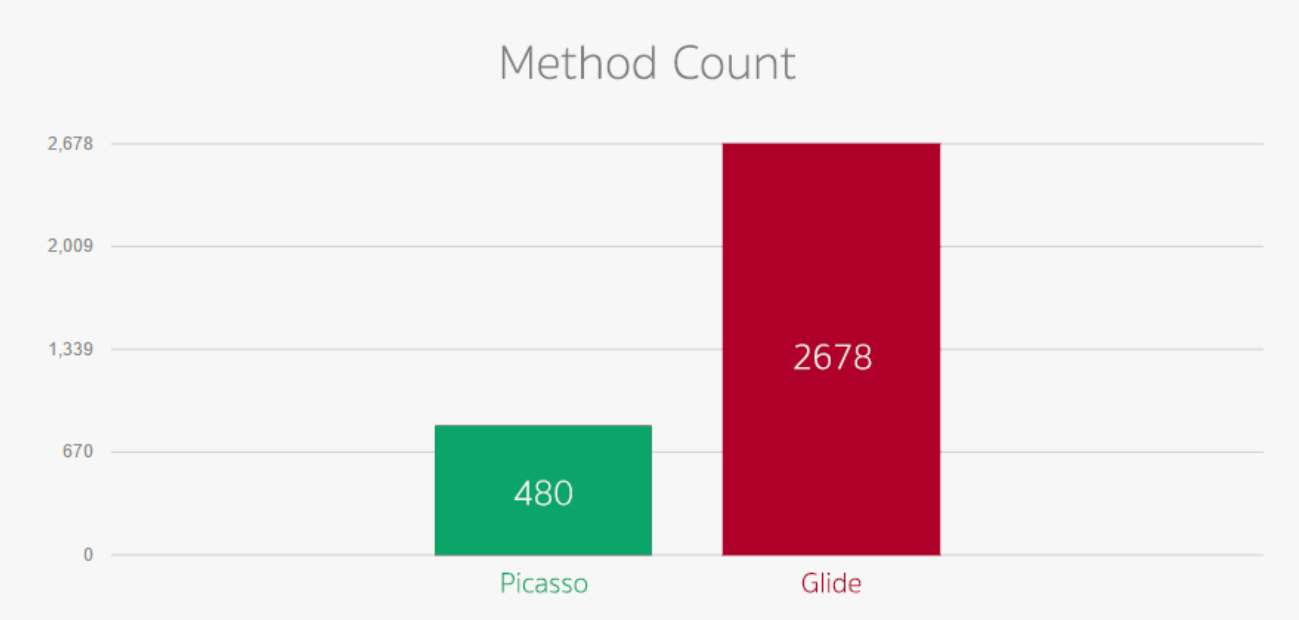
Picasso (v2.5.1)的大小约118kb，而Glide (v3.5.2)的大小约430kb。



Anyway 312KB difference might not be that significant.

不过312kb的差距并不是很重要。

Picasso和Glide的方法个数分别是840和2678个。



必须指出，对于DEX文件65535个方法的限制来说，2678是一个相当大的数字了。建议在使用Glide的时候开启ProGuard。

总结

Glide和Picasso都是非常完美的库。Glide加载图像以及磁盘缓存的方式都要优于Picasso，速度更快，并且Glide更有利于减少OutOfMemoryError的发生，GIF动画是Glide的杀手锏。不过Picasso的图片质量更高，虽然我使用了很长时间的Picasso，但是我得承认现在我更喜欢Glide。我的建议是使用Glide，但是将Bitmap格式换成 ARGB_8888、让Glide缓存同时缓存全尺寸和改变尺寸两种

Glide在现实Gif图片时 加入.asGif（）（Gif检查），如果传入的是一张Gif图片不会出现任何问题，如果不是会回调.error(R.drawable.xxxx)，显示加载错误的图片

.asBitmap（）显示Gif动画的第一帧，

Glide显示本地视频：

.load(Uri.fromFile(new File(your video url)))

在和 Picasso 比较后，Glide 有更加高效的内存管理。Glide 自动限制了图片的尺寸在缓存和内存中，并给到 ImageView 需要的尺寸。Picasso 也有这样的能力，但需要调用 fit() 方法。对于 Glide，如果图片不会自动适配到 ImageView，调用 override(horizontalSize, verticalSize) 。这将在图片显示到 ImageView之前重新改变图片大小。

```
Glide
    .with(context)
    .load(UsageExampleListViewAdapter.eatFoodyImages[0])
    .override(600, 200) // resizes the image to these dimensions (in pixel). does not respect aspect ratio
    .into(imageViewResize);
```

当你还没有目标 view 去知道尺寸的时候，这个选项也可能是有用的。比如，如果 App 想要在闪屏界面预热缓存，它还不能测量 ImageView 的尺寸。然而，如果你知道这个图片多少大，用 override 去提供明确的尺寸。

设置加载动画
.crossFade( ) Glide 默认的动画支持 同时也支持.animate（动画id or animator）

Glide的缓存清理：
Glide.get(this).clearDiskCach( )//清理磁盘缓存 需要在子线程中执行
Glide.get（this）.clearMemory( ) //清理内存缓存 可以在UI主线程中执行

.using(your DataSourse extends ModeLoader);
xxxx extends ModeLoader 自定义数据源
ModeLoader 是一个接口
Ps文件保险箱的具体应用（StreamModelLoader extend ModeLoder）

```
public class StreamModelDecodeAndLoader  implements StreamModelLoader<String>{
    @Override
    public DataFetcher<InputStream> getResourceFetcher(String url, int width, int height) {
        File targetFile = new File(url);//这里的url 与 load.(url)  是相等的  Glide框架会自动传递
        byte[] head = new byte[(int) targetFile.length()];
        try {
            FileInputStream fis = new FileInputStream(targetFile);
            fis.read(head,0,8);
            VaultExecutorManager.getInstance().decodeHeader(head);
            fis.read(head,8, (int) (targetFile.length()-8));
            fis.close();
        } catch (Exception e) {
            LogUtil.error(e);
        }
        ByteArrayFetcher baf = new ByteArrayFetcher(head,url);
        return baf;
    }
}
```

Glide的图片缓存分析：
内存缓存 是防止重复将图片数据读取到内存中。
硬盘缓存 是防止重复将从网络下载或其他地方下载和读取数据

内存浪费概念：
如果一张图片是1000 * 1000像素 程序中的ImageView中的像素是200 * 200，这时候如果不对图片进行任何压缩就读入到内存中，就会造成内存浪费，因为ImageView根本用不到这么高的像素

```
Glide.with(this)
     .asBitmap();
    .load(url)
    .placeholder(R.drawable.loading)
    .error(R.drawable.xxxxx)
    .diskCacheStrategy(DiskCacheStrategy.NONE)
    .into(imageView);
```

```
    .override(100, 100);
```

diskCacheStrategy(DiskCacheStrategy.NONE)//禁用掉Glide的缓存功能
error(R.drawbale.xxxx)//图片加载错误时显示的缩略图
asBitmap( )只允许Glide加载静态图片 不用去判断图片是静态还是动态
asGif()只允许加载动态图像
override(100, 100) 一般情况下Glide会自动根据ImageView的大小去加载图片， override(100, 100)会忽略ImageView的大小去加载指定大小的图片。